Compressing Convolutional Neural Networks via Factorized Convolutional Filters

Tuanhui Li[1] Baoyuan Wu[2*] Yujiu Yang[1*] Yanbo Fan[2] Yong Zhang[2] Wei Liu[2]

[1]Graduate School at Shenzhen, Tsinghua University [2]Tencent AI Lab

lth17@mails.tsinghua.edu.cn, wubaoyuan1987@gmail.com,yang.yujiu@sz.tsinghua.edu.cn,
{fanyanbo0124,zhangyong201303}@gmail.com, wl2223@columbia.edu

Abstract

*This work studies the model compression for deep convolutional neural networks (CNNs) via filter pruning. The workflow of a traditional pruning consists of three sequential stages: pre-training the original model, selecting the pre-trained filters via ranking according to a manually designed criterion (*e.g.*, the norm of filters), and learning the remained filters via fine-tuning. Most existing works follow this pipeline and focus on designing different ranking criteria for filter selection. However, it is difficult to control the performance due to the separation of filter selection and filter learning. In this work, we propose to conduct filter selection and filter learning simultaneously, in a unified model. To this end, we define a factorized convolutional filter (FCF), consisting of a standard real-valued convolutional filter and a binary scalar, as well as a dot-product operator between them. We train a CNN model with factorized convolutional filters (CNN-FCF) by updating the standard filter using back-propagation, while updating the binary scalar using the alternating direction method of multipliers (ADMM) based optimization method. With this trained CNN-FCF model, we only keep the standard filters corresponding to the* 1*-valued scalars, while all other filters and all binary scalars are discarded, to obtain a compact CNN model. Extensive experiments on CIFAR-10 and ImageNet demonstrate the superiority of the proposed method over state-of-the-art filter pruning methods.*

1. Introduction

Many popular deep convolutional neural networks have emerged in recent years, *e.g.*, VGGNet [32] and ResNet [10], *etc*. These models show promising results on many visual tasks, such as image classification [18, 36, 37, 39], semantic segmentation [2, 26], object detection [6], object tracking [22, 43] or visual reasoning [34, 42]. However, the model sizes and computation complexities of these deep models also grow exponentially. For example, ResNet-152 [10] contains about 60 million parameters and 11.3 billion FLOPS, which precludes the application of these models to mobile systems. A feasible approach to tackle this difficulty is model compression, whose goal is to reduce the parameters while keeping the model performance as much as possible.

Many seminal works have been developed in the literature of model compression for deep convolutional neural networks. They can be generally partitioned to four categories, including pruning [8, 12, 21, 23, 27], low-rank factorization [16, 19, 45], weight quantification [24, 25] and compact network design [13, 28], respectively. In this work, we focus on the pruning approach, and we refer the readers to [4] for more details about other categories. Specifically, we focus on the filter-level pruning (filter pruning), which prunes the output channel of a filter tensor. A typical workflow of the filter pruning is demonstrated in Fig. 1 (top). It consists of three sequential stages, including the training of the original model, pruning filters according to a manually designed ranking criterion, and fine-tuning the model with remained filters. Many existing works focused on designing different ranking criteria. However, most of these criteria depend on the weights values themselves or the results (*e.g.*, the classification accuracy) of the pre-trained original model. A typical criterion is the assumption that the filters with small 'weight' norms have small contributions to the model, thus they can be pruned [21]. However, to the best of our knowledge, we have never found a rigorous verification of this assumption. What is more important, the manually designed ranking criterion only depends on the pre-trained original model, rather than the followed fine-tuning process of the pruned model. The efficacy of the ranking criterion

*indicates corresponding authors. This work was done when Tuanhui Li was an intern at Tencent AI Lab.

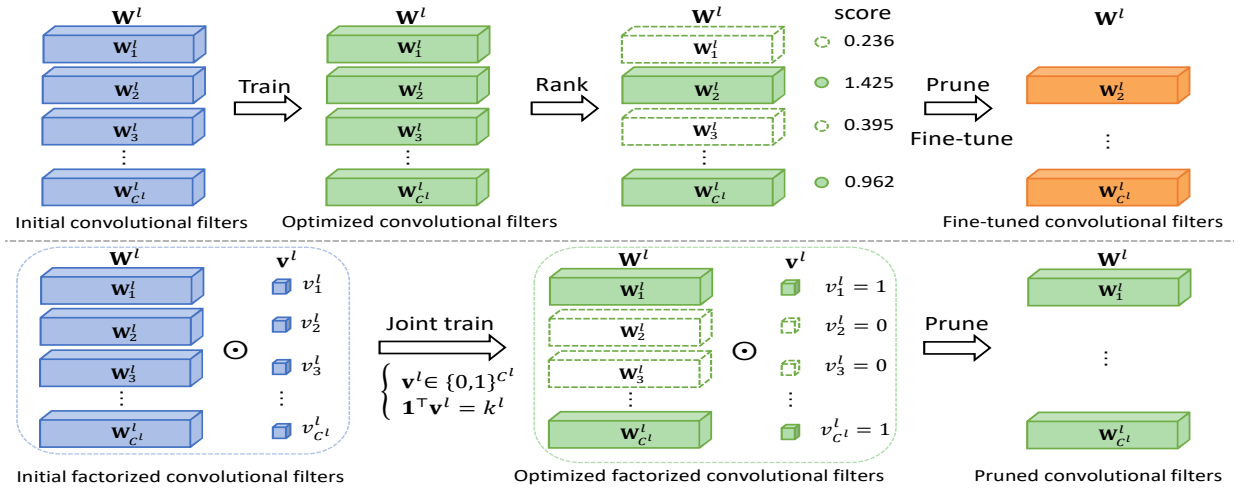

Figure 1. Overview of the workflow of filter pruning on layer l ($1 \leq l \leq L$), where the dotted green cubes indicate the pruned filters. **(Top)**: Traditional pruning consists of three sequential stages: pre-training, selecting filters according to a ranking criterion, and fine-tuning. **(Bottom)**: Our method conducts the filter learning and filter selection jointly, through training factorized convolutional filters.

can be only measured according to the fine-tuning result. It may take many iterations of pruning and fine-tuning to find a good ranking criterion.

In this work, we propose to conduct filter learning and filter selection jointly, in a unified optimization framework. To this end, we design a novel type of filter, dubbed *factorized convolutional filter* (FCF). As shown in Fig. 1 (bottom), a FCF consists of a standard convolutional filter $\mathbf{W}_i^l$ (l is the layer index and i is the filter index), and a binary scalar $v_i^l \in \{0, 1\}$, as well as a dot-product operator between them. After training the CNN model with factorized convolutional filters (CNN-FCF), the standard filters corresponding to 0-valued binary scalars and all binary scalars in other FCFs are directly abandoned, to construct a compact CNN model (see the green filters in Fig. 1 (bottom)). However, due to the binarity of v_i^l, the standard back-propagation with gradient descent algorithm cannot be directly adopted to train CNN-FCF. To tackle this difficulty, inspired by the general integer programming framework, *i.e.*, ℓ_p-Box ADMM [38], the binary vector $\mathbf{v}^l$ (including all binary scalars in layer l) is equivalently reformulated as a continuous vector, being subjected to the intersected constraint space between the box constraint and the ℓ_2-sphere constraint. Consequently, we propose a novel training algorithm for CNN-FCF by inserting the alternating direction method of multipliers (ADMM) [3] algorithm into the back-propagation framework. Specifically, $\mathbf{W}^l$ is updated by the standard gradient descent algorithm, while $\mathbf{v}^l$ is updated by the ADMM algorithm. We compress the popular ResNet models with different layers on two benchmark datasets, including CIFAR-10 [17] and ImageNet LSVRC-2012 [31] (ImageNet for clarity). Experimental results verify the competitive performance of the proposed method, compared to the state-of-the-art filter pruning methods.

The main contributions of this work are three-fold. **(1)** We propose to conduct filter learning and filter selection jointly in a unified optimization framework, through training CNNs with factorized convolutional filters. **(2)** We propose a novel algorithm to train CNN-FCF by inserting the ADMM algorithm into the standard gradient-based back-propagation framework. **(3)** Extensive experiments on popular ResNet models and benchmark datasets demonstrate the efficacy of the proposed method.

2. Related Work

In this section, we briefly review the pruning-based compressing approach, which can be generally partitioned to two levels, including weight-level and filter-level pruning.

Weight-level pruning (weight pruning for clarity) is an unstructured pruning method that prunes some entries in each filter. It was firstly proposed in optimal brain damage [20] and optimal brain surgeon [9], which pruned the weights according to the second order derivatives of the loss function. Recently, Han *et al.* [8] proposed to remove the weights with small values below the threshold. Guo *et al.* [7] proposed an interactive method, which is composed of pruning and splicing. Zhang *et al.* [44] formulated the pruning problem into a non-convex optimization problem and combined the weights with cardinality constraints. However, weight pruning can only produce a sparse network. Consequently, the memory and computational cost of the compressed model are not significantly reduced.

Filter-level pruning (filter pruning for clarity) is a structured pruning method that prunes the whole channel of filters. Thus, it can reduce more parameters and computation costs than the weight pruning method. **(1)** Some works calculate the importance score for each filter according to a manually designed evaluation criterion. A commonly used metric is the norm of filters [11, 21], based on the assump-

tion that filters with small norms do not contribute much to the model performance. Similar to filter norm, Hu *et al.* [14] proposed to rank the filters with the average percentage of zeros of the corresponding output feature maps. Yu *et al.* [41] proposed to obtain the importance score of other layers via back-propagating the filter scores of the final response layer. **(2)** Regularization constraints are also usually used in many studies for pruning. Some studies introduced group lasso with filters to directly derive sparse filters [1, 35]. Liu *et al.* [40] and Ye *et al.* [23] imposed ℓ_1 constraint to the scaling factors of BN layers, and the magnitudes of these scaling factors are used as the filter scores. Huang *et al.* [15] introduced the scaling factors with a ℓ_1 regularization for the selection of different micro-structures such as residual blocks. **(3)** Some works proposed to minimize the reconstruction error between the original model and the pruned model. He *et al.* [12] formulated the reconstruction error using lasso regression to retain the representative filters. Luo *et al.* [27] proposed to minimize the reconstruction error based on greedy search. Inspired by optimal brain damage, Dong *et al.* [5] performed Taylor expansion on the reconstruction error function, and determined the parameter importance according to the second order derivatives. Zhuang *et al.* [46] proposed to seek discriminative channels by minimizing both reconstruction error loss and additional loss. The main commonality of above related works is that the filter selection is somewhat separated with filter learning. In contrast, the proposed method conducts filter selection and filter learning simultaneously.

3. Model Compression

3.1. Convolutional Filter

Given training dataset consists of N samples $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, with $\mathbf{x}_i$ being input feature and $\mathbf{y}_i$ being ground-truth label of $\mathbf{x}_i$. Considering a CNN model with L layers, we use $\mathbf{W}^l \in \mathbb{R}^{C^l \times N^l \times W^l \times H^l}$ to represent the filters of the l-th convolutional layer, where (C^l, N^l, W^l, H^l) are the number of output channels, the number of input channels, the width of kernel and the height of kernel, respectively. The convolutional operation of layer l is formulated as

$$\mathbf{R}_{out}^l = \text{Conv}(\mathbf{R}_{in}^l, \mathbf{W}^l), \tag{1}$$

where Conv denotes the convolutional operation. $\mathbf{R}_{in}^l \in \mathbb{R}^{N \times N^l \times W_{in}^l \times H_{in}^l}$ and $\mathbf{R}_{out}^l \in \mathbb{R}^{N \times C^l \times W_{out}^l \times H_{out}^l}$ indicate the input and output responses of layer l, respectively.

3.2. Factorized Convolutional Filter

To facilitate the filter selection, we propose a novel filter, dubbed *factorized convolutional filter* (FCF), by associating a binary scalar $v_i^l \in \{0, 1\}$ to each filter $\mathbf{W}_i^l \in \mathbb{R}^{N^l \times W^l \times H^l}$ (l is layer index and i is filter index), through a dot-product operator. It is formulated as follows:

$$\mathbf{R}_{out}^l = \text{Conv}(\mathbf{R}_{in}^l, \mathbf{W}^l \odot \mathbf{v}^l), \tag{2}$$

where $\mathbf{v}^l = [\ldots; v_i^l; \ldots] \in \{0, 1\}^{C^l}$ and $\mathbf{W}^l \odot \mathbf{v}^l = [\ldots; \mathbf{W}_i^l \odot \mathbf{v}_i^l; \ldots] \in \mathbb{R}^{C^l \times N^l \times W^l \times H^l}$, with $\mathbf{W}_i^l \odot \mathbf{v}_i^l$ indicating that v_i^l is multiplied to every element in $\mathbf{W}_i^l$.

3.3. Training CNNs with Factorized Convolutional Filters

Joint training. In this section, we present how to conduct filter learning and filter selection jointly, based on the factorized convolutional filters. Let $\mathbf{W} = \{\mathbf{W}^l\}_{l=1}^L$, $\mathbf{v} = \{\mathbf{v}^l\}_{l=1}^L$, we denote $f(\mathbf{x}_i; \mathbf{W}, \mathbf{v})$ as the output probability of a CNN model with factorized convolutional filters (CNN-FCF). Then, the objective function of training CNN-FCF is formulated as follows:

$$\begin{aligned} &\arg\min_{\mathbf{W},\mathbf{v}} \quad \frac{1}{N}\sum_{i=1}^{N} \ell\big(\mathbf{y}_i, f(\mathbf{x}_i; \mathbf{W}, \mathbf{v})\big) \\ &\text{s.t. } \mathbf{1}^\top \mathbf{v}^l = k^l, \mathbf{v}^l \in \{0, 1\}^{C^l}, \ \forall\, l \in \{1, 2, ..., L\}, \end{aligned} \tag{3}$$

where $k^l \in \{1, 2, \ldots, C^l\}$ denotes the number of remained filters in layer l after pruning. The loss function ℓ can be specified by any loss function that could be used to train standard CNNs. In our experiments for image classification, we adopt the cross entropy loss. Due to the binary constraint on $\mathbf{v}$, Problem (3) cannot be directly optimized by continuous optimization algorithm (*e.g.*, the gradient-based back-propagation algorithm). Thus, we propose a novel continuous optimization algorithm, dubbed *back-propagation with ADMM*, as detailed in Section 4.

Filter pruning. Given a trained CNN-FCF model through optimizing Problem (3), one can obtain **1)** a compact CNN model by pruning the filters $\mathbf{W}_i^l$ corresponding to zero-valued v_i^l in each layer (see Fig. 1 (bottom)), or **2)** a sparse CNN model by setting zeros to the filters $\mathbf{W}_i^l$ corresponding to the zero-valued v_i^l. The choice of this two types of models depends on the model architecture and the pruning strategy, which will be detailed in Section 5.2.

4. Optimization

4.1. Continuous Reformulation

To tackle the binary constraint in Problem (3), we firstly transform the binary constraint to continuous constraints using the following technique proposed in [38], as follows:

$$\mathbf{v}^l \in \{0, 1\}^{C^l} \Leftrightarrow \mathbf{v}^l \in S_b \cap \mathbf{v}^l \in S_p, \tag{4}$$

where $S_b = [0, 1]^{C^l}$ denotes a box constraint, and $S_p = \left\{\mathbf{v}^l : \|\mathbf{v}^l - \frac{1}{2}\|_2^2 = \frac{C^l}{4}\right\}$ indicates a ℓ_2-sphere constraint. Furthermore, following the ℓ_p-Box ADMM algorithm [38], we introduce two additional variables to split the continuous constraints, so that they can be satisfied alternatively. Consequently, Problem (3) can be equivalently reformulated as

the following continuous problem:

$$\begin{aligned} \underset{\mathbf{W},\mathbf{v},\mathbf{z}_1,\mathbf{z}_2}{\arg\min} \quad & \frac{1}{N}\sum_{i=1}^{N}\ell\big(\mathbf{y}_i, f(\mathbf{x}_i;\mathbf{W},\mathbf{v})\big) \\ \text{s.t.} \quad & \mathbf{v}^l=\mathbf{z}_1^l, \quad \mathbf{v}^l=\mathbf{z}_2^l, \quad \mathbf{1}^\top\mathbf{z}_1^l=k^l, \\ & \mathbf{z}_1^l\in S_b, \quad \mathbf{z}_2^l\in S_p, \quad \forall\, l\in\{1,2,...,L\}, \end{aligned} \tag{5}$$

where $\mathbf{z}_1=\{\mathbf{z}_1^l\}_{l=1}^L$ and $\mathbf{z}_2=\{\mathbf{z}_2^l\}_{l=1}^L$. For clarity, hereafter we shorten $\frac{1}{N}\sum_{i=1}^{N}\ell\big(\mathbf{y}_i, f(\mathbf{x}_i;\mathbf{W},\mathbf{v})\big)$ as $\mathcal{L}(\mathbf{W},\mathbf{v})$.

4.2. Back-propagation with ADMM

Although Problem (5) is continuous, the back-propagation algorithm with the standard gradient-based optimizer (*e.g.*, stochastic gradient descent (SGD) [30]) can't be directly used to optimize the constrained problem. We propose a new algorithm by inserting the ADMM algorithm into the standard back-propagation training framework. Specifically, considering layer l, the parameters are updated by the following alternative steps:

- Given the fixed $\mathbf{v}^l$, the parameter $\mathbf{W}^l$ can be updated using gradient-based optimizer (see Section 4.2.1);
- Given the fixed $\mathbf{W}^l$, the variables $\mathbf{v}^l, \mathbf{z}_1^l, \mathbf{z}_2^l$ are updated by the ADMM algorithm (see Section 4.2.2).

This algorithm is outlined in Algorithm 1. To facilitate the following derivations of the proposed algorithm, we decompose the output function $f(\mathbf{x}_i;\mathbf{W},\mathbf{v})$ with respect to the parameters of layer l, as follows:

$$\begin{cases} f(\mathbf{x}_i;\mathbf{W},\mathbf{v}) = f_1\Big(\mathbf{R}_{out}^l;\{\mathbf{W}^j\}_{j=l+1}^L,\{\mathbf{v}^j\}_{j=l+1}^L\Big), \\ \mathbf{R}_{out}^l = \text{Conv}(\mathbf{R}_{in}^l, \mathbf{W}^l\odot\mathbf{v}^l), \\ \mathbf{R}_{in}^l = f_2\Big(\mathbf{x}_i;\{\mathbf{W}^j\}_{j=1}^{l-1},\{\mathbf{v}^j\}_{j=1}^{l-1}\Big), \end{cases} \tag{6}$$

where $\mathbf{R}_{in}^l$ and $\mathbf{R}_{out}^l$ denote the input and output response of layer l, respectively.

4.2.1 Given $\mathbf{v}^l$, Solving $\mathbf{W}^l$ Using Gradient Descent

$\mathbf{W}^l$ can be updated using the standard gradient descent,

$$\mathbf{W}^l = \mathbf{W}^l - \eta_{\mathbf{W}}\frac{\partial\mathcal{L}(\mathbf{W},\mathbf{v})}{\partial\mathbf{W}^l}, \tag{7}$$

where $\eta_{\mathbf{W}}$ denotes the learning rate. Using the chain rule and the decomposition in Eq. (6), we have

$$\begin{cases} \dfrac{\partial\mathcal{L}(\mathbf{W},\mathbf{v})}{\partial\mathbf{W}^l} = \dfrac{\partial\mathcal{L}}{\partial f_1}\times\dfrac{\partial f_1}{\partial\mathbf{R}_{out}^l}\times\dfrac{\partial\mathbf{R}_{out}^l}{\mathbf{W}^l}, \\ \dfrac{\partial\mathbf{R}_{out}^l}{\mathbf{W}^l} = \dfrac{\partial\mathbf{R}_{out}^l}{\partial\mathbf{W}^l\odot\mathbf{v}^l}\times\dfrac{\partial\mathbf{W}^l\odot\mathbf{v}^l}{\partial\mathbf{W}^l} = \dfrac{\partial\mathbf{R}_{out}^l}{\partial\mathbf{W}^l\odot\mathbf{v}^l}\times\mathbf{V}^l, \end{cases}$$

where $\mathbf{V}^l$ is expanded from $\mathbf{v}^l$, and has the same shape as $\mathbf{W}^l$. All the terms can be easily computed as did in training standard CNNs.

Algorithm 1 Back-propagation with ADMM

Input: Training data $\{\mathbf{x}_i,\mathbf{y}_i\}_{i=1}^N$ and initial $\mathbf{W}^l, \mathbf{v}^l, \forall\, l\in\{1,2,...,L\}$.
Output: $\hat{\mathbf{W}}^l$.

```
while not converged do
  for l = 1 to L do
    Given v^l, update W^l using gradient descent (see Section 4.2.1);
    Given W^l, update v^l using ADMM (see Section 4.2.2).
  end for
end while
if v_i^l = 1 then
  Ŵ_i^l = W_i^l (l is layer index and i is filter index).
end if
return Ŵ^l
```

4.2.2 Given $\mathbf{W}^l$, Solving $\mathbf{v}^l$ Using ADMM

With the fixed $\mathbf{W}^l$ and parameters of all other layers, Problem (5) with respect to $\mathbf{v}^l, \mathbf{z}_1^l, \mathbf{z}_2^l$ can be solved by the alternating direction method of multipliers (ADMM) [3] algorithm. Following the standard ADMM procedure, we firstly present the augmented Lagrangian function, as follows:

$$\begin{aligned} L(\mathbf{W}^l,\mathbf{v}^l,\mathbf{z}_1^l,\mathbf{z}_2^l,\mathbf{u}_1^l,\mathbf{u}_2^l) &= \mathcal{L}(\mathbf{W},\mathbf{v}) \\ &+ h_1(\mathbf{z}_1^l) + h_2(\mathbf{z}_2^l) + (\mathbf{u}_1^l)^\top(\mathbf{v}^l-\mathbf{z}_1^l) \\ &+ (\mathbf{u}_2^l)^\top(\mathbf{v}^l-\mathbf{z}_2^l) + \frac{\rho^l}{2}\Big[\|\mathbf{v}^l-\mathbf{z}_1^l\|_2^2 + \|\mathbf{v}^l-\mathbf{z}_2^l\|_2^2\Big], \end{aligned} \tag{8}$$

where $h_1(\mathbf{z}_1^l) = \mathbb{I}(\mathbf{z}_1^l \in S_b\cap\{\mathbf{z}_1^l:\mathbf{1}^\top\mathbf{z}_1^l=k^l\})$ and $h_2(\mathbf{z}_2^l)=\mathbb{I}(\mathbf{z}_2^l\in S_p)$ are indicator functions. $\mathbb{I}(a)=0$ if a is true, otherwise $\mathbb{I}(a)=\infty$. $\mathbf{u}_1^l, \mathbf{u}_2^l\in\mathbb{R}^{C^l}$ are dual variables, and $\rho^l>0$ is a penalty parameter. Then, in the following, we iteratively update $(\mathbf{z}_1^l,\mathbf{z}_2^l,\mathbf{v}^l)$ by minimizing the Lagrangian during each training iteration, and update $(\mathbf{u}_1^l,\mathbf{u}_2^l)$ by gradient ascent.

Update $(\mathbf{z}_1^l,\mathbf{z}_2^l)$: They are updated by solving the following sub-problems:

$$\begin{cases} \mathbf{z}_1^l = \arg\min_{\mathbf{z}_1^l\in S_c}\frac{1}{2}\|\mathbf{z}_1^l\|_2^2 - (\mathbf{v}^l+\frac{\mathbf{u}_1^l}{\rho^l})^\top\mathbf{z}_1^l, \\ \mathbf{z}_2^l = \arg\min_{\mathbf{z}_1^l\in S_p}\frac{1}{2}\|\mathbf{z}_2^l\|_2^2 - (\mathbf{v}^l+\frac{\mathbf{u}_2^l}{\rho^l})^\top\mathbf{z}_2^l, \end{cases} \tag{9}$$

where $S_c = S_b\cap\{\mathbf{z}_1^l:\mathbf{1}^\top\mathbf{z}_1^l=k^l\}$. The first sub-problem is a standard quadratic program (QP) problem, which can be globally optimized by any off-the-shelf QP solver. In our experiments, we adopt the OSQP solver [33]. The second sub-problem can be globally optimized by projecting the solution of the unconstrained problem onto the ℓ_2-sphere (*i.e.*, S_p), as presented in [38].

Update $\mathbf{v}^l$: Due to the loss term $\mathcal{L}(\mathbf{W},\mathbf{v})$, the sub-problem

with respect to $\mathbf{v}^l$ doesn't have a closed form solution. Instead, we adopt the gradient descent algorithm, as follows:

$$\mathbf{v}^l = \mathbf{v}^l - \eta_{\mathbf{v}} \frac{\partial L(\mathbf{W}^l, \mathbf{v}^l, \mathbf{z}_1^l, \mathbf{z}_2^l, \mathbf{u}_1^l, \mathbf{u}_2^l)}{\partial \mathbf{v}^l}, \tag{10}$$

where,

$$\begin{cases} \frac{\partial L}{\partial \mathbf{v}^l} = \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{v})}{\partial \mathbf{v}^l} + \frac{\partial \mathcal{C}}{\partial \mathbf{v}^l}, \\ \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{v})}{\partial \mathbf{v}^l} = \frac{\partial \mathcal{L}}{\partial f_1} \times \frac{\partial f_1}{\partial \mathbf{R}_{out}^l} \times \frac{\partial \mathbf{R}_{out}^l}{\mathbf{v}^l}, \\ \frac{\partial \mathbf{R}_{out}^l}{\mathbf{v}^l} = \frac{\partial \mathbf{R}_{out}^l}{\partial \mathbf{W}^l \odot \mathbf{v}^l} \times \frac{\partial \mathbf{W}^l \odot \mathbf{v}^l}{\partial \mathbf{v}^l} = \frac{\partial \mathbf{R}_{out}^l}{\partial \mathbf{W}^l \odot \mathbf{v}^l} \times \mathbf{W}^l, \\ \frac{\partial \mathcal{C}}{\partial \mathbf{v}^l} = \mathbf{u}_1^l + \mathbf{u}_2^l + \rho^l(2\mathbf{v}^l - \mathbf{z}_1^l - \mathbf{z}_2^l). \end{cases}$$

Update $(\mathbf{u}_1^l, \mathbf{u}_2^l)$:

$$\begin{cases} \mathbf{u}_1^l = \mathbf{u}_1^l + \rho^l(\mathbf{v}^l - \mathbf{z}_1^l), \\ \mathbf{u}_2^l = \mathbf{u}_2^l + \rho^l(\mathbf{v}^l - \mathbf{z}_2^l). \end{cases} \tag{11}$$

Besides, to accelerate the convergence process, we increase ρ^l by μ^l after each iteration, *i.e.*, $\rho^l \leftarrow \mu^l \rho^l$, and set an upper bound ρ_{max}^l to avoid early stopping. They will be specified in experiments.

4.2.3 Complexity Analysis

Here we analyze the computational complexity of layer l in Algorithm 1. The complexity of the forward $\mathbf{R}_{out}^l = \text{Conv}(\mathbf{R}_{in}^l, \mathbf{W}^l \odot \mathbf{v}^l)$ (see Eq. 2) is $O(NC^l N^l W^l H^l W_{in}^l H_{in}^l)$. The complexity of the backward $\frac{\partial \mathbf{R}_{out}^l}{\mathbf{W}^l}$ (see Section 4.2.1) is $\mathcal{O}(NC^l N^l W_{out}^l H_{out}^l W^l H^l)$. They are same with the complexities of forward and backward pass in standard convolutional layer. The additional complexity is mainly from the QP solver for $\mathbf{z}_1^l$, which is $O((C^l)^3)$. Considering the performance enhancement from the joint training, we believe that such an extra training cost is acceptable.

5. Experiments

5.1. Experimental Settings

Dataset. Our experiments are conducted on two benchmark datasets, including CIFAR-10 [17] and ImageNet [31]. CIFAR-10 has 50k training images and 10k validation images, which is annotated by 10 classes. ImageNet contains 1.28 million training images and 50k validation images of 1000 classes.

Models and compared methods. We evaluate the proposed method on the ResNet [10] architecture with different layers. On CIFAR-10, we compress ResNet-20, ResNet-32, ResNet-56 and ResNet-110, comparing with state-of-the-art methods, inlcuding SNLI [40], SFP [11], Pruning [21], NISP [41]. On ImageNet, we compress ResNet-34 and ResNet-50 to compare with NISP [41], SFP [11], Pruning [21], ThiNet [27], SSS [15] and Channel Pruning [12].

Evaluation metrics. We adopt three metrics to evaluate the compressed model, including Params.↓%, FLOPs↓%, and Acc.↓%. Params.↓% denotes the ratio of pruned parameters from the original model. FLOPs↓% represents the ratio of decreased computational cost compared to the original model. Acc.↓% indicates the reduced accuracy compared to the accuracy of the orginal model. A better compressing performance corresponds the higher Params.↓% and FLOPs↓%, while the lower Acc.↓%.

Implementation details. In the first stage, the joint training process of $\mathbf{W}$ and $\mathbf{v}$ using the proposed algorithm of back-propagation with ADMM for at most 30 epochs. The learning rates of $\eta_{\mathbf{W}}$ (see Eq. 7) and $\eta_{\mathbf{v}}$ (see Eq. 10) are initialized as 0.1, and they are divided by 10 after every 10 epochs. In terms of the hyper-parameters of the ADMM part (see Section 4.2.2), we adopt same settings for all layers. Specifically, for l-th layer, $\mathbf{v}^l$ is initialized as $\mathbf{1}$ to include all filters at the very beginning, $(\mathbf{u}_1^l, \mathbf{u}_2^l, \mathbf{z}_1^l, \mathbf{z}_2^l)$ are initialized as $\mathbf{0}$. On ImageNet, ρ^l is initialized as 0.001; μ^l, ρ_{max}^l and the batch-size are set as 1.001, 6 and 256, respectively. On CIFAR-10, ρ^l is initialized as 0.01; μ^l, ρ_{max}^l and the batch-size are set as 1.01, 6 and 128, respectively. Besides the maximum number of epochs (*i.e.*, 30), we also set another stopping criterion, *i.e.*, $\|\mathbf{v}^l - \mathbf{z}_1^l\|_2^2 \leqslant 10^{-4}$ and $\|\mathbf{v}^l - \mathbf{z}_2^l\|_2^2 \leqslant 10^{-4}, \forall\, l \in \{1, 2, \ldots, L\}$. Then, we binarize $\mathbf{v}^l$ to exactly $\mathbf{0}$ or $\mathbf{1}$. Theoretically speaking, if $\mathbf{v}^l$ converges exactly to $\mathbf{0}$ or $\mathbf{1}$, the trained CNN-FCF model will be the output. However, due to the numerical reason, there are still small changes on $\mathbf{v}^l$ after binarization, *i.e.*, $\mathbf{W}_i^l \odot |1 - v_i^l|$ or $\mathbf{W}_i^l \odot |v_i^l - 0|$. In Section 5.5, we will study the influence of these small differences. To alleviate the potential influence of these small differences, we also conduct fine-tuning on the compact CNN model in the second stage. In the fine-tuning process, we adopt SGD with 0.9 momentum, and the intial learning rate is 0.01. The weight decay factor is set as 0.0001. On ImageNet, we fine-tune for 90 epochs with the batch-size 256, and the learning rate is divided by 10 every 30 epochs. On CIFAR-10, we fine-tune for 150 epochs with the batch-size 128, and the learning rate is divided by 10 every 60 epochs. All experiments are implemented using PyTorch [29].

5.2. Pruning Strategies and Pruning Ratios

Pruning strategies. In the experiments, we find that many existing filter pruning works (*e.g.*, [21], [41]) not only prune the output channels of the parameter tensor, but also prune the input channels. Compared to the pruning only on the output channels, this strategy can remove useless parameters and reduce unnecessary computations, leading to higher Param.↓% and FLOPs↓%. In the following, we present brief definitions of these two pruning strategies.

- **Single-channel pruning.** As shown in Fig. 2 (top), single-channel pruning only prunes the output channels in $\mathbf{W}^l$ and $\mathbf{W}^{l+1}$.
- **Pair-channel pruning.** However, as shown in Fig. 2 (bottom), when a filter in $\mathbf{W}^l$ is pruned, the corresponding response in $\mathbf{R}^l_{out}$ will be removed (see the green dashed part of $\mathbf{R}^l_{out}$ in Fig. 2). Consequently, the filter at the input channel of $\mathbf{W}^{l+1}$ that corresponds to this removed response should also be removed (see the cube with green dashed lines in Fig. 2 (bottom)).

Remark. As most existing works change the model architecture via pruning the filters before fine-tuning, they have to adopt the pair-channel pruning strategy to match the output channel of $\mathbf{W}^l$ and the input channel of $\mathbf{W}^{l+1}$. Besides, if there is a short-cut connection (*e.g.*, ResNet), then the last layer in each block cannot be pruned in many existing works. Besides, as most entries in the pruned filters are likely to be non-zero, the fine-tuning is always required to recover the performance after pruning.

In contrast, as we don't change the model architecture when training CNN-FCF, all above limitations don't exist in our method. Our pruning method is much more flexible than existing works. The training process of CNN-FCF can be seen as the single-channel pruning strategy. However, given a trained CNN-FCF model, we can also adopt the pair-channel pruning strategy to set filters' parameters w.r.t. the pruned input channels to **0** for each layer. Given a trained CNN-FCF model, we can obtain a compact or sparse CNN models depending on the original model structures: **1)** If there is no short-cut, we can directly prune the filter $\mathbf{W}^l_i$ corresponding to zero-valued v^l_i to obtain a compact CNN model. One further point should be pointed out is that if there is a batch-normalization (BN) layer (*e.g.*, ResNet), then the mean and variance values of BN will be modified after pruning, leading to the change of responses. Thus, it also requires fine-tuning to resume the model performance, as did in most existing works. **2)** If there is a short-cut and we also perform filter pruning (*i.e.*, set a pruning ratio using the cardinality constraint) for the last layer in each block, then we cannot directly prune the filters as did above. The reason is that the remained filters in two layers connected by the short-cut cannot be aligned. Instead, we obtain a sparse CNN model, by setting zeros to the convolutional filters corresponding to zero-valued v^l_i. When using this sparse CNN model for inference, we design a *squeeze-convolution-expand* procedure. Specifically, for each sparse convolutional tensor, we firstly squeeze it by removing zero-valued filters to obtain a small dense tensor, then conduct convolution using this small tensor, finally expand the obtained feature map to the original shape by filling zeros. In practice, the computational costs of the squeeze and expand steps are negligible compared to the convolution step.

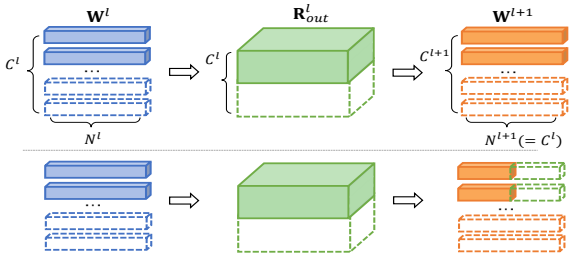

Figure 2. **(Top)**: Single-channel pruning only prunes the output channels of $\mathbf{W}^l$ and $\mathbf{W}^{l+1}$. **(Bottom)**: Pair-channel pruning not only prunes the output channels of $\mathbf{W}^{l+1}$, but also prunes its input channels corresponding to the pruned output channels in $\mathbf{W}^l$.

Model	Method	Params.↓%	FLOPs↓%	Ref.%[1]	Acc.↓%
ResNet-20	SNLI [40]	37.22	–[2]	92.00	1.10
	SFP [11]	–	**42.20**	92.20	1.37
	CNN-FCF	**42.75**	41.60	92.20	**1.07**
	SNLI [40]	67.83	–	92.00	3.20
	CNN-FCF	**68.44**	**68.91**	92.20	**2.67**
ResNet-32	SFP [11]	–	41.50	92.63	0.55
	CNN-FCF	**42.71**	**42.21**	92.43	**0.25**
	CNN-FCF	69.46	70.21	92.43	1.69
ResNet-56	Pruning-A [21]	9.40	10.40	93.04	-0.06
	Pruning-B [21]	13.70	27.60	93.04	-0.02
	SFP [11]	–	41.10	93.59	-0.19
	NISP [41]	42.60	**43.61**	–	0.03
	CNN-FCF	**43.09**	42.78	93.14	**-0.24**[3]
	CNN-FCF	69.74	70.90	93.14	1.22
ResNet-110	Pruning-A [21]	2.30	15.90	93.53	0.02
	Pruning-B [21]	32.40	38.60	93.53	0.23
	SFP [11]	–	40.80	93.68	**-0.18**
	NISP [41]	**43.25**	**43.78**	–	0.18
	CNN-FCF	43.19	43.08	93.58	-0.09
	CNN-FCF	69.51	70.81	93.58	0.62

[1] Ref. denotes the accuracy of the pre-trained original model.
[2] – means that the metric value is not reported in the compared method.
[3] Negative value means the pruned model accuracy is higher than the Ref.

Table 1. Comparison results on CIFAR-10.

Pruning ratios. We also notice that some existing works (*e.g.*, ThiNet [27] and NISP [41]) adopted the same pruning ratio for all pruned convolutional layers, especially when there is short-cut in the model (*e.g.*, ResNet). However, we believe that different layers of one CNN model have different contributions to the model performance. Thus, the ratios of parameter redundancy of different layers should be different. We will evaluate the influence of the same or different pruning ratios in experiments (see Section 5.4).

5.3. Comparisons with State-of-the-art Methods

In this section, to compare with state-of-the-art methods (*e.g.*, ThiNet [27] and NISP [41]), our method which denoted as CNN-FCF also adopts the pair-channel pruning strategy and same pruning ratios for all layers.

Results on CIFAR-10. As shown in Table 1, we compress ResNet-20, ResNet-32, ResNet-56 and ResNet-110 with two pruning ratios, including 43% and 69%. On ResNet-20, at the pruning ratio of about 43%, our method gives the smallest accuracy loss (*i.e.*, 1.07% Acc.↓%). In con-

Model	Method	Params.↓%	FLOPs↓%	Top1 Ref.%	Top1↓%	Top5 Ref.%	Top5↓%
ResNet-34	Pruning [21]	10.80	24.20	73.23	1.06	–	–
	NISP [41]	**27.14**	**27.32**	–	0.28	–	-
	CNN-FCF	27.05	26.83	73.30	**-0.25**	91.42	**-0.08**
	SFP [11]	–	41.10	73.92	2.09	91.62	1.29
	NISP [41]	**43.68**	**43.76**	–	0.92	–	–
	CNN-FCF	42.19	41.38	73.30	**0.51**	91.42	**0.47**
	CNN-FCF	55.80	54.87	73.30	1.97	91.42	1.22
	CNN-FCF	67.24	66.05	73.30	3.59	91.42	2.11
ResNet-50	SSS [15]	27.06	**31.08**	76.12	1.94	92.86	0.95
	NISP [41]	**27.12**	27.31	–	0.21	–	–
	CNN-FCF	26.70	29.24	76.15	**-0.35**	92.87	**-0.26**
	ThiNet [27]	33.72	36.79	75.30	1.27	92.20	**0.09**
	SFP [11]	–	41.80	76.15	1.54	92.87	0.81
	NISP [41]	**43.82**	44.01	–	0.89	–	–
	Channel pruning [12]	–	**50.00**	–	–	92.20	1.40
	CNN-FCF	42.41	46.05	76.15	**0.47**	92.87	0.19
	CNN-FCF	52.52	57.10	76.15	1.60	92.87	0.69
	CNN-FCF	61.01	66.17	76.15	2.62	92.87	1.37

Table 2. Comparison results on ImageNet. "Top1 Ref.%" denotes the top1 accuracy of the original model.

trast, SNLI [40] gives the slightly higher accuracy loss (*i.e.*, 1.1% Acc.↓%), while the pruned ratio of parameters is less than ours, *i.e.*, 37.22 vs. 42.75 of Params.↓%. The pruned ratio of parameters by SFP [11] is similar with ours, but its accuracy loss is higher, *i.e.*, 1.37 vs. 1.07 of Acc.↓%. At the pruning ratio of about 69%, SNLI also performs worse than our method, with the higher accuracy loss, *i.e.*, 3.2 vs. 2.67 of Acc.↓%. On other ResNet models with different layers, our method also show very competitive performance, compared to other methods. Moreover, when comparing the compressing performance of our method on ResNet models with different layers, an interesting observation is that the accuracy loss decreases along with the increasing of the number of layers, at the same pruning ratio. Specifically, at the pruning ratio of about 43%, the values of Acc.↓% of our method are $1.07, 0.25, -0.24, -0.09$, on ResNet models with 20, 32, 56 and 110 layers, respectively; at the pruning ratio of about 69%, the corresponding values are $2.67, 1.69, 1.22, 0.62$. It demonstrates that the ResNet models with deeper layers have the larger redundancy on the image classification task on CIFAR-10.

Results on ImageNet. As shown in Table 2, we compress ResNet-34 and ResNet-50 with four pruning ratios, including about $27\%, 43\%, 55\%, 67\%$. We present the losses on both top1 and top5 accuracy. On ResNet-34, at the pruning ratio of about 27%, the Top1↓% value of our method is -0.25, while that of NISP [41] is 0.28. At the pruning ratio of about 43%, Param.↓% and FLOPs↓% of our method are slightly lower than NISP, but our Top1↓% is also lower than that of NISP, *i.e.*, 0.51 vs. 0.92. At the pruning ratios of about 55% and 67%, the values of Top1↓% of our method are only 1.97 and 3.59, respectively; the values of Top5↓% of our method are only 1.22 and 2.11, respectively. On ResNet-50, our method also shows very competitive performance, compared to state-of-the-art methods. Moreover, similar to the results on CIFAR-10, we observe that there is also a large proportion of redundant parameters in ResNet, even for the image classification task on ImageNet.

5.4. Analysis of Pruning Strategies and Ratios

As demonstrated in Section 5.2, here we evaluate the influence if we allocate distinct pruning ratios for different layers. Inspired by [21, 23], we try a simple allocation of pruning ratios as follows: **(1)** Calculating the score of each filter using the ℓ_1-norm, *i.e.*, $s_i^l = |\mathbf{W}_i^l|_1$; **(2)** Storing all filter scores s_i^l to a vector $\mathbf{s}$, and sorting $\mathbf{s}$ in an ascending order; **(3)** Assume that the overall pruning ratio is p, then the number of filters to be pruned is $N_p = p|\mathbf{s}|$. We set the N_p-th value of $\mathbf{s}$ as a global threshold, denoted as $\mathbf{s}_p$; **(4)** Counting the number of s_i^l exceeding $\mathbf{s}_p$ in each layer, which is recorded as k_i^l (see Problem (3)). For comparison, we also evaluate the identical ratio for all layers. After training the CNN-FCF with same or distinct pruning ratios, we can choose single-channel or pair-channel pruning strategy before fine-tuning. Thus, there are four settings, including identical-ratio with single-channel, identical-ratio with pair-channel, distinct-ratio with single-channel and distinct-ratio with pair-channel, respectively. The results are shown in Table 3. Given the same level of Params.↓% on same dataset and model, the accuracy loss of distinct ratios is always much lower than that of identical ratio. It demonstrates that distinct ratios obtain much better compressing performance. However, the FLOPs↓% value of distinct ratios is always lower than that of identical ratio. The reason is that in the setting of distinct ratios, the ratio allocation strategy described above assigns more pruning ratios to the higher layers (*i.e.*, closer to the output layer), where the #FLOPs is smaller than that of the lower layers, as the corresponding feature maps become smaller. In the future, we will explore other allocation strategies to achieve more reductions of #FLOPs. In comparison between single-channel and pair-channel pruning, given the same level of Params.↓%, the same dataset and model, pair-channel pruning always gives lower accuracy loss. The reason is that there are some useless parameters in the single-channel pruning. In summary, the setting of training the CNN-FCF model with distinct pruning ratio, and with the pair-channel pruning strategy for fine-tuning, obtains the best compressing performance.

5.5. Analysis of Different Compressing Stages

In this section, we analyze the influence of different stages of a compression procedure of the proposed method, including the joint optimization of $\mathbf{W}$ and $\mathbf{v}$ in the CNN-FCF model, pruning the filter corresponding to the 0-valued v_i^l, and fine-tuning. Besides, we also compare with two baselines, including pruning the pre-trained CNN model randomly (denoted as *Random* in Table 4 and Fig. 3), and pruning using the ranking (*i.e.*, ranking the filters in each layer using the ℓ_1 norm in descent order, and pruning the lower-ranked filters with a pre-defined ratio. This setting is denoted as *Ranking* in Table 4 and Fig. 3). We evaluate above methods for compressing the ResNet-56 model

(a) single-channel pruning

Dataset	Model	Params.↓%	FLOPs↓%	Top1↓%	Top5↓%
CIFAR-10	ResNet56-I	**43.18**	**43.15**	0.42	–
	ResNet56-D	42.66	32.02	**-0.14**	–
	ResNet56-I	69.14	**71.09**	2.33	–
	ResNet56-D	**70.17**	53.08	**0.75**	–
ImageNet	ResNet34-I	**27.33**	**26.96**	0.59	0.36
	ResNet34-D	27.23	16.31	**-0.03**	**0.16**
	ResNet34-I	42.42	**42.13**	2.25	1.40
	ResNet34-D	**43.71**	22.88	**0.87**	**0.50**

(b) pair-channel pruning

Dataset	Model	Params.↓%	FLOPs↓%	Top1↓%	Top5↓%
CIFAR-10	ResNet56-I	**43.09**	**42.78**	**-0.24**	–
	ResNet56-D	41.97	33.99	-0.16	–
	ResNet56-I	**69.74**	**70.90**	1.22	–
	ResNet56-D	69.57	55.67	**0.64**	–
ImageNet	ResNet34-I	27.05	**26.83**	-0.25	-0.08
	ResNet34-D	**27.38**	22.21	**-0.28**	**-0.08**
	ResNet34-I	42.19	**41.38**	0.51	0.47
	ResNet34-D	**43.72**	28.42	**0.39**	**0.28**

Table 3. Results of CNN-FCF with identical and distinct pruning ratios. ResNet56-I denotes pruning ResNet-56 using identical pruning ratio for each layer, while ResNet56-D indicates distinct ratios for different layers.

Method	Ref.%[1]	Optim.%[2]	Pruning%[3]	Fine-tuning%[4]
Random	93.14	–	10.00	91.14
Ranking	93.14	–	10.00	92.09
CNN-FCF-S	93.14	91.92	91.44	92.35
CNN-FCF-P	93.14	92.36	30.17	92.89

[1] Ref.% indicates the accuracy of the original model.
[2] Optim.% denotes the accuracy of the optimized CNN-FCF model.
[3] Pruning% denotes the accuracy after pruning.
[4] Fine-tuning% denotes the accuracy after fine-tuning.

Table 4. Compressing Performance of ResNet-56 on CIFAR-10, with the pruning ratio of 45% parameters.

on CIFAR-10, with the overall pruning ratio of 45% parameters and the identical pruning ratio of all layers. The joint optimization of CNN-FCF takes at most 150 epochs. As shown in Table 4, all methods starts from the same checkpoint of ResNet-56, of which the accuracy is 93.14%. In terms of the Random method, after pruning 45% parameters from the pre-trained model, the accuracy drops to 10%; after fine-tuning, the accuracy resumes to 91.14%. In terms of the Ranking method, the accuracy also drops to 10% after pruning, and it achieves 92.09% after fine-tuning. It demonstrates that the fine-tuning is crucial for these two baselines, and Ranking keeps better filters than Random. CNN-FCF-S denotes our method with single-channel pruning. After the joint optimization of $\mathbf{W}$ and $\mathbf{v}$ in CNN-FCF, the accuracy achieves 91.92%, which is very close to 93.14% of the pre-trained CNN model. After the single-channel pruning, there is a slight drop of accuracy, *i.e.*, 0.48%. As analyzed in *Implementation details* of Section 5.1, it is due to the numerical difference between the converged $\mathbf{v}$ and the binarized $\mathbf{v}$. After fine-tuning, the accuracy resumes to 92.35%. CNN-FCF-P denotes our method with pair-channel pruning. After the joint optimization, the accuracy achieves 92.36%, which is higher than that of CNN-FCF-S. The reason is that although the overall pruning ratios are same (*i.e.*, 45%), the filter cardinality (*i.e.*, k^l) of CNN-FCF-P is larger than that of CNN-FCF-S, as it will prune more parameters in the pruning stage. However, after pair-channel pruning, the accuracy significantly drops to 30.17%. The reason is that the pruning on the input channels of filters significantly changes the trained CNN-FCF model. But the accuracy achieves 92.89% after fine-tuning, which is highest among all compared methods. We also show the fine-tuning curves of above four methods in Fig. 3. The above comparisons demonstrate that: **(1)** The proposed compressing method based on CNN-FCF performs better than baselines; **(2)** The joint training based on CNN-FCF produces a very good compressed model, even without fine-tuning; **(3)** When adopting the pair-channel pruning strategy, fine-tuning is useful to resume the model performance.

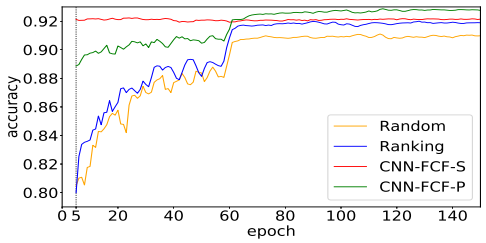

Figure 3. The accuracy curves of fine-tuning of four methods. As the accuracies of all methods excluding CNN-FCF-S are small in the first 5 epochs, we ignore them to highlight the accuracy differences in final epochs. Please refer to Section 5.5 for details.

6. Conclusion

This work presented a novel model compression method for deep CNNs, of which the core idea is conducting filter learning and filter selection jointly. To this end, we defined a novel type of filters, dubbed factorized convolutional filter (FCF), consisting of a standard convolutional filter and a binary scalar, as well as a dot-product operator between them. To train a CNN model with FCF, we proposed to insert the ADMM algorithm into the back-propagation framework via gradient descent. Given a trained CNN-FCF model, a compact or a sparse standard CNN model can be obtained by only keeping the standard filters corresponding to the 1-valued scalars. Experiments on compressing ResNet models demonstrate the superiority of the proposed method over state-of-the-art filter pruning methods.

Acknowledgement The involvements of Yujiu Yang and Tuanhui Li in this work were supported in part by the National Key Research and Development Program of China (No.2018YFB1601102), and Shenzhen special fund for the strategic development of emerging industries (No.JCYJ20170412170118573).

References

[1] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016. 3

[2] Linchao Bao, Baoyuan Wu, and Wei Liu. Cnn in mrf: Video object segmentation via inference in a cnn-based higher-order spatio-temporal mrf. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5977–5986, 2018. 1

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011. 2, 4

[4] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19(1):64–77, 2018. 1

[5] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017. 3

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 1

[7] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems*, pages 1379–1387, 2016. 2

[8] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016. 1, 2

[9] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 164–171, 1993. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1, 5

[11] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *International Joint Conferences on Artificial Intelligence*, 2018. 2, 5, 6, 7

[12] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of IEEE International Conference on Computer Vision*, 2017. 1, 3, 5, 7

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[14] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 3

[15] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. *arXiv preprint arXiv:1707.01213*, 2017. 3, 5, 7

[16] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 1

[17] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 2, 5

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. 1

[19] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 1

[20] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605, 1990. 2

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. 1, 2, 5, 6, 7

[22] Xin Li, Chao Ma, Baoyuan Wu, Zhenyu He, and Ming-Hsuan Yang. Target-aware deep tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 1

[23] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of IEEE International Conference on Computer Vision*, pages 2755–2763, 2017. 1, 3, 7

[24] Zechun Liu, Wenhan Luo, Baoyuan Wu, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Binarizing deep network towards real-network performance. *arXiv preprint arXiv:1811.01335*, 2018. 1

[25] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European Conference on Computer Vision*, pages 722–737, 2018. 1

[26] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 1

[27] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. Thinet: Pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 1, 3, 5, 6, 7

[28] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *Proceedings of the European Conference on Computer Vision*, 2018. 1

[29] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration, 2017. 5

[30] Herbert Robbins and Sutton Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985. 4

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 2, 5

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 1

[33] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *ArXiv e-prints*, Nov. 2017. 4

[34] Kaihua Tang, Hanwang Zhang, Baoyuan Wu, Wenhan Luo, and Wei Liu. Learning to compose dynamic tree structures for visual contexts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 1

[35] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016. 3

[36] Baoyuan Wu, Weidong Chen, Yanbo Fan, Yong Zhang, Jinlong Hou, Junzhou Huang, Wei Liu, and Tong Zhang. Tencent ml-images: A large-scale multi-label image database for visual representation learning. *arXiv preprint arXiv:1901.01703*, 2019. 1

[37] Baoyuan Wu, Weidong Chen, Peng Sun, Wei Liu, Bernard Ghanem, and Siwei Lyu. Tagging like humans: Diverse and distinct image annotation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7967–7975, 2018. 1

[38] Baoyuan Wu and Bernard Ghanem. lp-box admm: A versatile framework for integer programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018. 2, 3, 4

[39] Baoyuan Wu, Fan Jia, Wei Liu, and Bernard Ghanem. Diverse image annotation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2559–2567, 2017. 1

[40] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. 3, 5, 6, 7

[41] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 3, 5, 6, 7

[42] Hanwang Zhang, Zawlin Kyaw, Shih-Fu Chang, and Tat-Seng Chua. Visual translation embedding network for visual relation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5532–5540, 2017. 1

[43] Tianzhu Zhang, Bernard Ghanem, Si Liu, and Narendra Ahuja. Robust visual tracking via multi-task sparse learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2042–2049, 2012. 1

[44] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision*, 2018. 2

[45] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, 2016. 1

[46] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, 2018. 3